TABLE OF CONTENTS (CR18)

LOGICAL VIEW REPORT

Logical View

VISA

This package contains the classes for communicating with instruments through the VISA standard.

INSTRInterface

This class is the concrete base class for any class wishing to support the INSTR interface in the VISA standard. Note, this class assumes access to the Rogue Wave Tools.h++ library and the National Instruments VISA library.

Derived from VISAInterface

Public Methods:

INSTRInterface(VISAResourceManager& manager, ViRsrc address, const RWCString& interfaceName, ViAccessMode accessMode, ViUInt32 timeOut);
Constructs a new INSTRInterface at the given address, with the interface name, access mode, and time out value.

INSTRInterface();
Constructs a new uninitialized INSTRInterface

INSTRInterface(const VISAInterface& vi);
Constructs a new INSTRInterface from a previous INSTRInterface.

virtual ~INSTRInterface();
Destroys the INSTRInterface.

const VISAInterface& operator=(const VISAInterface& vi);
Sets self to a shallow copy of vi.

bool operator==(const INSTRInterface& i);
True if the two interfaces are the same. The interfaces are the same if they have the same address.

bool operator!=(const INSTRInterface& i);
True if the two interfaces are different. The interfaces are different if they do not have the same address.

RWCString address();
Returns the address of the interface.

RWCString getInterfaceName();
Returns the name of the type of interface.

unsigned int getInterfaceNumber();
Returns the board number of the interface.

unsigned int getInterfaceType();
Returns the type of interface.

unsigned int getIoProtocol();
Returns the io protocol of interface.

unsigned long getQueueLength();
Returns the length of the event queue for the interface.

unsigned long getTimeOut();
Returns the time out value for the interface.

const RWCString& name();
Returns the name of the interface.

bool sendEnd();
Returns true if the interface is going to asset an END during the transfer of the last byte of the buffer.

void setIoProtocol(unsigned int protocol);
Sets the io protocol of interface.

void setQueueLength(unsigned long length);
Sets the length of the event queue for the interface.

bool setSendEnd(bool yesOrNo);
SEtst the interface to assert END during the transfer of the last byte of the buffer, if yesOrNo is true.

void setTimeOut(unsigned long timeOut);
Sets the time out value for the interface.

GPIBInterface

This class is the concrete base class for any class wishing to support the GPIB interface in the VISA standard. Also, the GPIBInterface supports the VISA io completion and wait for service events. This class operates under the assumption that there will only be one io operation taking place at any one time. Note, this class assumes access to the Rogue Wave Tools.h++ and National Instruments VISA library.

Derived from INSTRInterface

Public Methods:

GPIBInterface(VISAResourceManager& manager, ViRsrc address, const RWCString& interfaceName, ViAccessMode accessMode, ViUInt32 timeOut);
Constructs a new GPIBInterface at the given address, with the interface name, access mode, and time out value.

GPIBInterface();
Constructs a new uninitialized GPIBInterface

GPIBInterface(const VISAInterface& vi);
Constructs a new GPIBInterface from a previous GPIBInterface.

virtual ~GPIBInterface();
Destroys the GPIBInterface.

const GPIBInterface& operator=(const GPIBInterface& gi);
Sets self to a shallow copy of gi.

bool operator==(const GPIBInterface& gi);
True if the two interfaces are the same. The interfaces are the same if they have the same address.

bool operator!=(const GPIBInterface& gi);
True if the two interfaces are different. The interfaces are different if they do not have the same address.

RWCString address();
Returns the address of the interface.

void enableTermCharacter(bool yesOrNo);
If yesOrNo is set to true the terminating character is sent after an io write operation.

RWCString getInterfaceName();
Returns the name of the type of interface.

unsigned int getInterfaceNumber();
Returns the board number of the interface.

unsigned int getInterfaceType();
Returns the type of interface.

unsigned int getIoProtocol();
Returns the io protocol of interface.

unsigned int getPrimaryAddress();
Returns primary GPIB address for the interface.

unsigned long getQueueLength();
Returns the length of the event queue for the interface.

bool getRepeatAddressing();
Returns true if the interface is set to use repeat addressing before any io operation.

unsigned int getSecondaryAddress();
Returns the secondary GPIB address for the interface.

char getTermCharacter();
Returns the io termination character.

unsigned long getTimeOut();
Returns the time out value for the interface.

bool getUnaddress();
Returns true if the interface is set to unaddress after an io operation.

void killRead();
Kills the current read operation.

void killWrite();
Kills the current write operation.

const RWCString& name();
Returns the name of the interface.

bool read(RWCString& buffer, ViUInt32 timeOut = VI_TMO_INFINITE);
Reads data from the interface and stores it in the buffer. Note, that the data read is appended to the buffer and does not overwrite what is already there. The default behavior is for the read call to block. Specifying a timeout value in milliseconds can change this behavior.

bool sendEnd();
Returns true if the interface is going to asset an END during the transfer of the last byte of the buffer.

void setIoProtocol(unsigned int protocol);
Sets the io protocol of interface.

void setQueueLength(unsigned long length);
Sets the length of the event queue for the interface.

void setRepeatAddressing(bool yesOrNo);
If yesOrNo is true, the interface is set to use repeat addressing.

bool setSendEnd(bool yesOrNo);
SEtst the interface to assert END during the transfer of the last byte of the buffer, if yesOrNo is true.

void setTermCharacter(char termChar);
Sets the io termination character to the termChar.

void setTimeOut(unsigned long timeOut);
Sets the time out value for the interface.

void setUnaddress(bool yesOrNo);
If yesOrNo the inteface is set to unaddress.

unsigned short int statusByte();
Returns the status byte.

bool trigger();
Asserts a GPIB software trigger.

bool waitForSRQ(short int& sb, ViUInt32 timeOut = VI_TMO_INFINITE);
Waits for an SRQ to be asserted on the bus and the status byte retrieved. The default behavior is for this call to block. Specifying a timeout value in milliseconds can change this behavior.

bool write(const RWCString& buffer, ViUInt32 timeOut = VI_TMO_INFINITE);
Writes the buffer to the interface. The default behavior is for the write call to block. Specifying a timeout value in milliseconds can change this behavior.

bool write(char* buffer, unsigned long numberOfBytes, ViUInt32 timeOut = VI_TMO_INFINITE);
Writes the buffer to the interface. The default behavior is for the write call to block. Specifying a timeout value in milliseconds can change this behavior.

VISAError

This class is the exception class any class that inherits from the VISAInterface. Note, this class assumes access to the Rogue Wave Tools.h++ library and the National Instruments VISA library.

Public Methods:

VISAError(const RWCString& myErrorMsg, long errorCode);
Constructs a new VISAError with a user message and VISA error code.

VISAError(long errorCode);
Constructs a new VISAError with the given VISA error code.

long errorCode();
Returns the VISA error code.

const RWCString& msg();
Returns the user message.

const RWCString& visaError();
Returns test form of the error code.

RWCString visaMsg();
Returns the VISA error message.

VISAResourceManager

This class initializes the VISA subsystem and must be created before any VISA objects are used. Only one VISAResourceManager should be created per thread. Also, the thread safety of this class is unknown. Note, this class assumes access to the Rogue Wave Tools.h++ library and National Instruments VISA library.

Public Methods:

VISAResourceManager0;
Constructs a new VISAResourceManager.

~VISAResourceManager0;
Destroys the VISAResourceManager.

ViSession operator()0;
Returns the VISA session for the resource manager.

VISAInterface

This class is the concrete base class for any class wishing to support instrument communications via the VISA standard. Note, this class assumes access to the Rogue Wave Tools.h++ library and the National Instruments VISA library.

Public Methods:

VISAInterface(VISAResourceManager& manager, ViRsrc address, const RWCString& interfaceName, ViAccessMode accessMode, ViUInt32 timeOut);
Constructs a new VISAInterface at the given address, with the interface name, access mode, and time out value.

VISAInterface();
Constructs a new uninitialized VISAInterface

VISAInterface(const VISAInterface& vi);
Constructs a new VISAInterface from a previous VISAInterface.

virtual ~VISAInterface();
Destroys the VISAInterface.

const INSTRInterface& operator=(const INSTRInterface& i);
Sets self to a shallow copy of i.

bool operator==(const VISAInterface& vi);
True if the two interfaces are the same. The interfaces are the same if they have the same address.

bool operator!=(const VISAInterface& vi);
True if the two interfaces are different. The interfaces are different if they do not have the same address.

RWCString address0;
Returns the address of the interface.

unsigned long getQueueLength0;
Returns the length of the event queue for the interface.

unsigned long getTimeOut0;
Returns the time out value for the interface.

const RWCString& name0;
Returns the name of the interface.

void setQueueLength(unsigned long length);
Sets the length of the event queue for the interface.

void setTimeOut(unsigned long timeOut);
Sets the time out value for the interface.

Protected Methods:

ViSession session();
Returns the VISA session associated with the interface.

SRSDS345

SRSDS345

This class is used for communications with the SRSDS345 function generator.

Derived from IPulseWave
Derived from ISinWave

Public Methods:

SRSDS345(VISAResourceManager& rm, const RWCString& address);
Creates an SRSDS345 at the given address.

~SRSDS345();
Destroys an SRSDS345.

void pulseWaveClear();
Clears the instrument making the pulsel wave.

void pulseWaveCommit();
Commits the pulse wave to the device making the wave.

void pulseWaveSetup(const RWCString& initialVoltage, const RWCString& pulseVoltage, const RWCString& delayTime, const RWCString& riseTime, const RWCString& fallTime, const RWCString& pulseTime, const RWCString& period);
Sets up the pulse wave.

void pulseWaveTrigger();
Triggers the pulse wave.

void pulseWaveWait(bool shouldWait = true);
Sets the device making the pulse wave to wait for an external trigger.

void sinWaveClear();
Clears the instrument making the sine wave.

void sinWaveCommit();
Commits the sine wave to the device making the wave.

void sinWaveSetup(const RWCString& offset, const RWCString& amplitude, const RWCString& frequency, const RWCString& delay, const RWCString& alpha, const RWCString& phase);
Sets up the sine wave.

void sinWaveTrigger();
Triggers the sine wave.

void sinWaveWait(bool shouldWait = true);
Sets the device making the sine wave to wait for an external trigger.

Measurement Request Broker

MRBActuator

This class abstracts all the instrument interfaces. It allows uniform access to any interface.

Public Methods:

MRBActuator(MRBDevice* pDevice);
Constructs an MRBActuator with the given MRBDevice.

~MRBActuator();
Destroys the MRBActuator.

void clear();
Clears the device in the actuator.

void commit();
Commits the device in the actuator.

MRBDevice* device();
Returns the device in the actuator.

bool increment();
Increments the actuator one step. Will return false when the actuator is incremented beyond its final value.

void reset();
Resets the actuator to its initial state.

void setup();
Sets up the device contained in the actuator.

void wait(bool wait = true);
Sets the device in the actuator to wait for external trigger.

MRBDevice

This class represents a device in circuit. It holds its location as well as all the properties associated with the device. Also, it contains the interface to access the instrument associated with the device.

Public Methods:

MRBDevice(const RWCString& name, int type, int subType);
Creates a new device with given name, type, and subtype.

MRBDevice(const MRBDevice& device);
Creates a new MRBDevice that is a copy of device.

MRBDevice();
Creates an empty MRBDevice.

~MRBDevice();
Destroys an MRBDevice.

operator==(const MRBDevice& testDevice);
Compare self to passed device.

RWCString& operator[](int property);
Returns the property at this location.

RWCString& at(int property);
Returns the property at this location.

bool getLocation(int wire, int& row, int& column);
Retrieves the matrix switch location of the given wire on the device.

ICurrentBias* iCurrentBias();
Returns an ICurrentBias pointer.

ICurrentData* iCurrentData();
Returns an ICurrentData pointer.

ICurrentSweep* iCurrentSweep();
Returns an ICurrentSweep pointer.

IOscope* iOscope();
Returns an IOscope pointer.

IPulseWave iPulseWave();
Returns an IPulseWave pointer.

ISinWave* iSinWave();
Returns an ISinWave pointer.

IVoltageBias* iVoltageBias();
Returns an IVoltageBias pointer.

IVoltageData* iVoltageData();
Returns an IVoltageData pointer.

iVoltageSetup();
Returns an IVoltageSetup pointer.

const RWCString& resourceName();
Name of the resource being encapsulated by this device.

void setColumn(int wire, int column);
Sets the column location of the given wire.

void setRow(int wire, int row);
Sets the row location for the given wire.

int subType();
Returns the subtype.

void takeResource(MRBDevice* pDevice);
Takes some of the properties from the passed device and sets them in the current device.

void takeRowsFrom(MRBDevice& device);
Takes rows from passed device and sets the rows in self.

int type();
Returns the type.

MRBCompiler

This class compiles a SPICE net list in to a list of MRBDevices called an MRBCircuit.

Public Methods:

MRBCompiler(const RWCString& lexTableName, const RWCString& yaccTableName);
Creates a new compiler.

~MRBCompiler();
Destroys an MRBCompiler.

bool compile(RWCString& netList, RWTPtrDlist<MRBCircuit>& circuits, RWCString& errorString);
Compiles the net list in to a list of MRBCircuits. All errors are placed in the error string.

MRBResource

This class encapsulates an MRBResource. An MRBResource represents a real life instrument or device, with all the interfaces it supports.

Public Methods:

~MRBResource();
Destroys an new MRBResource.

MRBResource();
Creates an new MRBResource.

void addDevice(MRBDevice* pDevice);
Adds a device to the resource

MRBDevice* getDevice(int type, int subtype);
Returns the device corresponding to the type and subtype.

bool supports(int type, int subtype);
Returns true if the resource supports the type and subtype of an interface.

MRBMeasurement

This class encapsulates a measurement as performed by the Measurement Request Broker.

Public Methods:

MRBMeasurement(RWTPtrDlist<MRBActuator>& deviceList, RWTPtrDlist<MRBActuator>& dataList, RWTPtrDlist<MRBResource> resourceList, RWTPtrDlist<MRBDevice>& nonActuatorList);
Creates a new MRBMeasurement.

~MRBMeasurement();
Destroys an MRBMeasurement.

void connect(IBigMatrix* pMatrixSwitch, RWCString& info);
Connects all the devices in the MRBMeasurement.

RWCString measure();
Does the measurement stored in the MRBMeasurement.

MRBCircuit

This class represents a circuit. It is a collection of MRBDevices.

MRBResourceAllocator

This class converts a MRBCircuit in to an MRBMeasurement.

Public Methods:

MRBResourceAllocator(const RWCString& nonSources, const RWCString& sources);
Creates an new MRBResourceAllocator.

~MRBResourceAllocator();
This destroys an MRBResourceAllocator.

bool allocate(MRBCircuit* circuit, MRBMeasurement*& pMeasurement, RWCString& errorString);
This allocates resource turning an MRBCircuit in to an MRBMeasurement.

deallocate(MRBMeasurement*& pMeasurement);
This deallocates the resources used in the MRBMeasurement.

InstrumentInterfaces

This package is set of instrument interfaces. Any instrument wishing to support a particular kind of functionality will inheirit from the interfaces in this package.

ICurrentSweep

This is the base interface for any instrument wishing to support current sweeps.

Public Methods:

virtual void currentSweepClear() = 0;
Clears the instrument performing the current sweep.

virtual void currentSweepCommit() = 0;
Commits the current sweep to the device performing the sweep.

virtual void currentSweepSetup(const RWCString& start, const RWCString& stop, const RWCString& increment, const RWCString& type, const RWCString& compliance) = 0;
Sets up the current sweep.

virtual void currentSweepTrigger() = 0;
Triggers the current sweep.

virtual void currentSweepWait(bool shouldWait = true) = 0;
Sets the device performing the current sweep to wait for an external trigger.

IVoltageSweep

This is the base interface for any instrument wishing to support voltage sweeps.

Public Methods:

virtual void voltageSweepClear() = 0;
Clears the instrument performing the voltage sweep.

virtual void voltageSweepCommit() = 0;
Commits the voltage sweep to the device performing the sweep.

virtual void voltageSweepSetup(const RWCString& start, const RWCString& stop, const RWCString& increment, const RWCString& type, const RWCString& compliance) = 0;
Sets up the voltage sweep.

virtual void voltageSweepTrigger() = 0;
Triggers the voltage sweep.

virtual void voltageSweepWait(bool shouldWait = true) = 0;
Sets the device performing the voltage sweep to wait for an external trigger.

ICurrentBias

This is the base interface for any instrument wishing to support current bias.

Public Methods:

virtual void currentBiasClear() = 0;
Clears the instrument performing the current bias.

virtual void currentBiasCommit() = 0;
Commits the current bias to the device performing the sweep.

virtual void currentBiasSetup(const RWCString& value, const RWCString& compliance) = 0;
Sets up the current bias.

virtual void currentBiasTrigger() = 0;
Triggers the current bias.

virtual void currentBiasWait(bool shouldWait = true) = 0;
Sets the device performing the current bias to wait for an external trigger.

IVoltageBias

This is the base interface for any instrument wishing to support voltage bias.

Public Methods:

virtual void voltageBiasClear() = 0;
Clears the instrument performing the voltage bias.

virtual void voltageBiasCommit() = 0;
Commits the voltage bias to the device performing the sweep.

virtual void voltageBiasSetup(const RWCString& value, const RWCString& compliance) = 0;
Sets up the voltage bias.

virtual void voltageBiasTrigger() = 0;
Triggers the voltage bias.

virtual void voltageBiasWait(bool shouldWait = true) = 0;
Sets the device performing the voltage bias to wait for an external trigger.

IVoltageData

This is the base interface for any instrument wishing to support voltmeters.

Public Methods:

virtual RWCString voltageData() = 0;
Returns the measured data.

virtual void voltageDataClear() = 0;
Clears the instrument performing the voltage measurement.

virtual void voltageDataCommit() = 0;
Commits the setup to the device performing the measurement.

virtual void voltageDataSetup(int node) = 0;
Sets up the voltage measurement.

ICurrentData

This is the base interface for any instrument wishing to support ammeters.

Public Methods:

virtual RWCString currentData() = 0;
Returns the measured data.

virtual void currentDataClear() = 0;
Clears the instrument performing the current measurement.

virtual void currentDataCommit() = 0;
Commits the setup to the device performing the measurement.

virtual void currentDataSetup(int node) = 0;
Sets up the voltage current.

ISinWave

This is the base interface for any instrument wishing to support sine waves.

Public Methods:

virtual void sinWaveClear() = 0;
Clears the instrument making the sine wave.

virtual void sinWaveCommit() = 0;
Commits the sine wave to the device making the wave.

virtual void sinWaveSetup(const RWCString& offset, const RWCString& amplitude, const RWCString& frequency, const RWCString& delay, const RWCString& alpha, const RWCString& phase) = 0;
Sets up the sine wave.

virtual void sinWaveTrigger() = 0;
Triggers the sine wave.

virtual void sinWaveWait(bool shouldWait = true) = 0;
Sets the device making the sine wave to wait for an external trigger.

IExpWave

This is the base interface for any instrument wishing to support exponetial waves.

Public Methods:

virtual void expWaveClear() = 0;
Clears the instrument making the exponential wave.

virtual void expWaveCommit() = 0;
Commits the exponetial wave to the device making the wave.

virtual void expWaveSetup(const RWCString& initialVoltage, const RWCString& pulseVoltage, const RWCString& riseDelayTime, const RWCString& riseTimeConst, const RWCString& fallDelayTime, const RWCString& fallTimeConst) = 0;
Sets up the exponential wave.

virtual void expWaveTrigger() = 0;
Triggers the exponential wave.

virtual void expWaveWait(bool shouldWait = true) = 0;
Sets the device making the exponential wave to wait for an external trigger.

IPulseWave

This is the base interface for any instrument wishing to support pulse waves.

Public Methods:

virtual void pulseWaveClear() = 0;
Clears the instrument making the pulsel wave.

virtual void pulseWaveCommit() = 0;
Commits the pulse wave to the device making the wave.

virtual void pulseWaveSetup(const RWCString& initialVoltage, const RWCString& pulseVoltage, const RWCString& delayTime, const RWCString& riseTime, const RWCString& fallTime, const RWCString& pulseTime, const RWCString& period) = 0;
Sets up the pulse wave.

virtual void pulseWaveTrigger() = 0;
Triggers the pulse wave.

virtual void pulseWaveWait(bool shouldWait = true) = 0;
Sets the device making the pulse wave to wait for an external trigger.

IOscope

This is the base interface for any instrument wishing to support oscilliscopes.

Public Methods:

virtual RWCString oscopeData() = 0;
Returns the measured data.

virtual void oscopeDataClear() = 0;
Clears the instrument performing the oscilliscope measurement.

virtual void oscopeDataCommit() = 0;
Commits the setup to the device performing the measurement.

virtual void oscopeDataSetup(int node, const RWCString& timeBase, const RWCString& verticalRange) = 0;
Sets up the voltage measurement.

ISmallMatrix

This class is the base interface for any instrument to be a matrix switch.

Public Methods:

virtual void close(unsigned int row, unsigned int column) = 0;
Closes a connection in the switch.

unsigned int columns();
Returns the number of columns in the switch.

virtual void free(unsigned int row) = 0;
Frees the given row in the switch.

virtual void freeAll() = 0;
Frees all connections in the switch.

virtual void open(unsigned int row, unsigned int column) = 0;
Opens a connection in the switch.

virtual unsigned int rows() = 0;
Returns the number of rows in the switch.

IBigMatrix

This class is the base interface for the full matrix switch.

Public Methods:

void add(ISmallMatrix*& pSmallSwitch);
Adds a small matrix switch to the large matrix switch.

virtual void close(unsigned int row, unsigned int column) = 0;
Closes a connection in the switch.

virtual unsigned int columns() = 0;
Returns the number of columns in the switch.

virtual void free(unsigned int row) = 0;
Frees the given row in the switch.

virtual void freeAll() = 0;
Frees all connections in the switch.

virtual void open(unsigned int row, unsigned int column) = 0;
Opens a connection in the switch.

virtual unsigned int rows() = 0;
Returns the number of rows in the switch.

HP4142B

This class is used for communications with the HP4124B.

Public Methods:

HP4142B(VISAResourceManager& rm, const RWCString& address);
Creates an HP4124B at the given address.

~HP4142B(VISAResourceManager& rm, const RWCString& address);
Destroys an HP4124B.

void clear();
Clears the HP4142B.

void commit();
Writes all commands out to the instrument.

RWCString getData(int channel);
Returns data for the given channel from the HP4142B.

void giveCommand(int channel, const RWCString& command);
Gives the command string for the given channel to the HP4142B.

void wait(bool shouldWait);
If shouldWait is true, then the HP4142B is set to trigger from an external source.

HP41421B

This class represents the HP41421B Source Measurement Unit cartridge for the HP4142B. It proxies the HP4142B.

Derived from ICurrentBias
Derived from ICurrentData
Derived from ICurrentSweep
Derived from IVoltageBias
Derived from IVoltageData
Derived from IVoltageSweep

Public Methods:

HP41421B(HP4142B& server, int channel);
This constructs an HP41421B for the given channel for the given HP4142B.

~HP41421B();
Destroys the HP41421B.

void currentBiasClear();
Clears the instrument performing the current bias.

void currentBiasCommit();
Commits the current bias to the device performing the sweep.

void currentBiasSetup(const RWCString& value, const RWCString& compliance);
Sets up the current bias.

void currentBiasTrigger();
Triggers the current bias.

void currentBiasWait(bool shouldWait = true);
Sets the device performing the current bias to wait for an external trigger.

RWCString currentData();
Returns the measured data.

void currentDataClear0;
Clears the instrument performing the current measurement.

void currentDataCommit0;
Commits the setup to the device performing the measurement.

void currentDataSetup(int node);
Sets up the voltage current.

void currentSweepClear0;
Clears the instrument performing the current sweep.

void currentSweepCommit0;
Commits the current sweep to the device performing the sweep.

void currentSweepSetup(const RWCString& start, const RWCString& stop, const RWCString& increment, const RWCString& type, const RWCString& compliance);
Sets up the current sweep.

void currentSweepTrigger0;
Triggers the current sweep.

void currentSweepWait(bool shouldWait = true);
Sets the device performing the current sweep to wait for an external trigger.

void voltageBiasClear0;
Clears the instrument performing the voltage bias.

void voltageBiasCommit0;
Commits the voltage bias to the device performing the sweep.

void voltageBiasSetup(const RWCString& value, const RWCString& compliance);
Sets up thevoltage bias.

void voltageBiasTrigger0;
Triggers the voltage bias.

void voltageBiasWait(bool shouldWait = true);
Sets the device performing the voltage bias to wait for an external trigger.

RWCString voltageData();
Returns the measured data.

void voltageDataClear0;
Clears the instrument performing the voltage measurement.

virtual void voltageDataCommit0 = 0;
Commits the setup to the device performing the measurement.

void voltageDataSetup(int node);
Sets up the voltage measurement.

void voltageSweepClear0;
Clears the instrument performing the voltage sweep.

void voltageSweepCommit0;
Commits the voltage sweep to the device performing the sweep.

virtual void voltageSweepSetup(const RWCString& start, const RWCString& stop, const RWCString& increment, const RWCString& type, const RWCString& compliance) = 0;
Sets up the voltage sweep.

void voltageSweepTrigger0;
Triggers the voltage sweep.

void voltageSweepWait(bool shouldWait = true);
Sets the device performing the voltage sweep to wait for an external trigger.

HP41424A

This class represents the HP41424A Voltage Source/ Voltage Measurement cartridge for the HP4142B. It proxies the HP4142B.

Derived from IVoltageBias
Derived from IVoltageData
Derived from IVoltageSweep

Public Methods:

HP41424A(HP4142B& server, int channel);
Constructs an HP41424A for the given channel for the given HP4142B.

~HP41424A();
Destroys an HP41424A.

void voltageBiasClear();
Clears the instrument performing the voltage bias.

void voltageBiasCommit();
Commits the voltage bias to the device performing the sweep.

void voltageBiasSetup(const RWCString& value, const RWCString& compliance);
Sets up the voltage bias.

void voltageBiasTrigger();
Triggers the voltage bias.

void voltageBiasWait(bool shouldWait = true);
Sets the device performing the voltage bias to wait for an external trigger.

RWCString voltageData();
Returns the measured data.

void voltageDataClear();
Clears the instrument performing the voltage measurement.

virtual void voltageDataCommit() = 0;
Commits the setup to the device performing the measurement.

void voltageDataSetup(int node);
Sets up the voltage measurement.

void voltageSweepClear();
Clears the instrument performing the voltage sweep.

void voltageSweepCommit();
Commits the voltage sweep to the device performing the sweep.

virtual void voltageSweepSetup(const RWCString& start, const RWCString& stop, const RWCString& increment, const RWCString& type, const RWCString& compliance) = 0;
Sets up the voltage sweep.

void voltageSweepTrigger();
Triggers the voltage sweep.

void voltageSweepWait(bool shouldWait = true);
Sets the device performing the voltage sweep to wait for an external trigger.

HPE1467A

HPE1467A

This class is the instrument driver for the HPE1467A VXI matrix switch.

Derived from ISmallMatrix

Public Methods:

HPE1467A(const RWCString& address);
Creates an HPE1467A at the given address.

~HPE1467A();
Destroys an HPE1467A.

void close(unsigned int row, unsigned int column);
Closes a connection in the switch.

unsigned int columns();
Returns the number of columns in the switch.

void free(unsigned int row);
Frees the given row in the switch.

void freeAll();
Frees all connections in the switch.

void open(unsigned int row, unsigned int column);
Opens a connection in the switch.

unsigned int rows();
Returns the number of rows in the switch.

HP54600B

HP54600B

This class is used for communications with the HP54600B.

Public Methods:

HP54600B(VISAResourceManager& rm, const RWCString& address);
This creates an HP54600B at the given address.

~HP54600B();
Destroys an HP54600B.

void clear();
Clears the HP4142B.

void commit();
Writes all commands out to the instrument.

RWCString getData(int channel);
Returns data for the given channel from the HP54600B.

void sendCommand(int channel, const RWCString& command);
Gives the command string for the given channel to the HP54600B.

HP54600BChannel

This class represents a channel of the HP54600B. It proxies the HP54600B.

Derived from IOscope

Public Methods:

HP54600BChannel(int channel, HP54600B& server);
Creates a HP54600BChannel for the given channel on the given HP54600B. This class proxies the HP54600B.

~HP54600BChannel();
Destroys the HP54600BChannel.

RWCString oscopeData();
Returns the measured data.

void oscopeDataClear();
Clears the instrument performing the oscilloscope measurement.

void oscopeDataCommit();
Commits the setup to the device performing the measurement.

void oscopeDataSetup(int node, const RWCString& timeBase, const RWCString& verticalRange);
Sets up the voltage measurement.

Instrument Drivers

PACKAGE STRUCTURE

Logical View

VISA
Instrument Interfaces
Instrument Drivers

HP4142B

HP54600B

SRSDS345

HPE1467A

Measurement Request Broker

```
void main()
{
   VISAResourceManager rm;

   //create the matrix switch
   ViSession s = rm();
   ISmallMatrix *pLower = NULL;
   ISmallMatrix *pUpper = NULL;
   IBigMatrix *pBig = NULL;

   CoInitializeEx(0, COINIT_MULTITHREADED);
   if (viOpenDefaultRM(&s) == VI_SUCCESS) cout <<"got the session\n";

   HRESULT hr = ::CoCreateInstance(CLSID_HPE1467A, NULL,
CLSCTX_INPROC_SERVER, IID_ISmallMatrix, (void**)&pLower);
   if (SUCCEEDED(hr))
   {
      cout << "got the lower interface\n";

      cout << "trying to create instance of the instrument\n";
      hr = pLower->Create(&s, "VXI0::120::INSTR", NULL,NULL);
      if (SUCCEEDED(hr)) cout << "VXI0::120::INSTR\n";
   }
   hr = ::CoCreateInstance(CLSID_HPE1467A, NULL, CLSCTX_INPROC_SERVER,
IID_ISmallMatrix, (void**)&pUpper);
   if (SUCCEEDED(hr))
   {
      cout << "got the upper interface\n";

      cout << "trying to create instance of the instrument\n";
      hr = pUpper->Create(&s, "VXI0::121::INSTR", NULL,NULL);
      if (SUCCEEDED(hr)) cout << "VXI0::121::INSTR\n";
   }
   hr = ::CoCreateInstance(CLSID_BigMatrix, NULL, CLSCTX_INPROC_SERVER,
IID_IBigMatrix, (void**)&pBig);
   if (SUCCEEDED(hr))
   {
      UINT x,y;

      cout << "got the big interface\n";

      cout << "trying to insert switches\n";
      pBig->Add(&pLower);
      pBig->Add(&pUpper);
      pBig->GetColumns(&x);
      pBig->GetLines(&y);

      cout << x << " " << y << endl;
   }

   RWCString lexTable("c:\\stephan\\marble\\server\\spice.dfa");
   RWCString yaccTable("c:\\stephan\\marble\\server\\spice.llr");
   MRBCompiler compiler(lexTable.data(), yaccTable.data());
   MRBResourceAllocator
resourceAllocator("c:\\stephan\\marble\\server\\debug\\nonsources.txt",
"c:\\stephan\\marble\\server\\debug\\sources.txt");
```

```
   //create the sock
   RWWinSockInfo sockInfo;

   //create the sock listener
   RWSocketListener serverSocket(RWSockAddr("inet lochaber.eecs.uic.edu
6969"));

   //output file
   RWCString outDirectory = "c:\\stephan\\niif\\serverlog\\";
   RWCString outFileName = "c:\\stephan\\niif\\serverlog\\";
   RWDate date;
   RWCString dt = date.asString().remove(2,1);
   dt = dt.remove(4,1);
   RWCString currentFile = dt + ".txt";
   outFileName =  outFileName + currentFile;
   ofstream logFile(outFileName, ios::app);

   for (;;)
   {
      try
      {
         RWSocketPortal sockPortal = serverSocket();
         RWTime startTime;
         RWTimer timer;
         timer.start();
         RWPortalIStream netIn(sockPortal);
         RWPortalOStream netOut(sockPortal);
         RWInetAddr inetAddr = sockPortal.socket().getpeername();
         cout << sockPortal.socket().getpeername() << endl;

         strstream temp;
         temp << sockPortal.socket().getpeername() << endl;
         RWCString tempClient;
         tempClient.readLine(temp);
         RWCRegexp address("[0-9]+\\.[0-9]+\\.[0-9]+\\.[0-9]+");
         RWCString clientAddress = tempClient(address);

         //get the port
         int clientReturnPort = 0;
         netIn >> clientReturnPort;

         RWCString userInfo;
         userInfo.readLine(netIn);
         cout << userInfo << endl;

         RWCString circuitString;
         circuitString.readLine(netIn);
         cout << circuitString << endl;

         //get the netlist
         RWCString netList;
         netList.readFile(netIn);

         //output the client data
         //cout << RWInetHost::addressAsString(inetAddr.host().address()) <<
endl;
         cout << "Client IP: " << clientAddress << endl;
```

```
        //cout << "Client Port: " << inetAddr.port() << endl;
        cout << "Client Data Port: " << clientReturnPort << endl;
        cout << "Net List" << endl;
        cout << netList << endl;

        //create error string and circuit list
        RWTPtrDlist<RWTPtrDlist<MRBDevice> > circuitList;
        RWCString error = "\0";
        RWCString data = "\0";
        RWCString connectionInfo = "\0";

        if (!compiler.compile(netList, circuitList, error))
        {
           cout << error << endl;
        }
        else
        {
           MRBMeasurement* pMeasurement;
           if (!resourceAllocator.allocate(circuitList.get(),
pMeasurement,error))
           {
              resourceAllocator.deallocate(pMeasurement);
              cout << error << endl;
           }
           else
           {
              try
              {
                 pMeasurement->connect(pBig, connectionInfo);
                 data = pMeasurement->measure();
                 cout << data << endl;
                 resourceAllocator.deallocate(pMeasurement);
              }
              catch (VISAError e)
              {
                 resourceAllocator.deallocate(pMeasurement);
                 cout << e.visaMsg();
                 error += e.visaMsg() + "$";
              }
           }
           circuitList.clear();
        }
        //try sending the data back
        try
        {
           RWPortal clientPortal =
RWSocketPortal(RWInetAddr(clientReturnPort, clientAddress));
           RWPortalOStream clientOut(clientPortal);

           if (error.isNull())
           {
              clientOut << data << endl;
           }
           else
           {
              clientOut << error << endl;
           }
```

```
        }
        catch( const RWxmsg& msg)
        {
           cerr << "Error: " << msg.why() << endl;
           error += msg.why();
        }
        RWTime stopTime;
        timer.stop();

        //get the current file
        RWDate tempDate;
        date = tempDate;
        dt = date.asString().remove(2,1);
        dt = dt.remove(4,1);
        RWCString tempFile = dt + ".txt";

        if (tempFile != currentFile)
        {
           logFile.close();
           outFileName = outDirectory + tempFile;
           logFile.open(outFileName, ios::app);
           currentFile = tempFile;
        }
        //write data to log file
        logFile << "Date: " << RWDate() << endl;
        logFile << "Start time: " << startTime << endl;
        logFile << "Stop time: " << stopTime << endl;
        logFile << "Total elapsed time: " << timer.elapsedTime() << endl;
        logFile << "Client IP: " << clientAddress << endl;
        //logFile << "Client Port: " << inetAddr.port() << endl;
        logFile << "Client Data Port: " << clientReturnPort << endl;
        logFile << "User info:" << endl;
        logFile << userInfo <<endl;
        logFile << "Circuit:" << endl;
        logFile << circuitString << endl;
        logFile << "Net List" << endl;
        logFile << netList;
        logFile << "NetListLength: " << netList.length() << endl;
        logFile << "Connection Info" << endl;
        logFile << connectionInfo << endl;
        logFile << "Data" << endl << data <<endl;
        logFile << "Error" << endl << error << endl << endl;
     }
     catch (const RWxmsg& msg)
     {
        cerr << "Error: " << msg.why() << endl;
        logFile << msg.why() << endl;
     }
  }
  CoUninitialize();
}
```